

DEVELOPING A CUSTOM TABLE WIDGET TO BE USED IN A SCREEN COMBINATION

Whenever we are presented with scrollable data on a mainframe host screen, it can be rendered using tables or other components in a screen combination event. However, whenever there is a special requirement in the display of data on the web page, we can develop our custom widgets to fulfill the requirement.

In this article we discuss a scenario in which we have data presented on the host screen as shown below:

```

SELECT CODE *----- NAME FIELD -----* ORG *CUST OR STORE NBR*
( ) 02 %DR. SMITH 000 00000000000000000066
( ) 01 ADAM SMITH 000 00000000000000000005
( ) E ALLEN SMITH 000 00000000000000000009
( ) E ALLEN SMITH 000 00000000000000000000

MORE.....
    
```

As we can see, each row has an input field at the beginning. We need to type an **X** into the input field of any record in order to view its details.

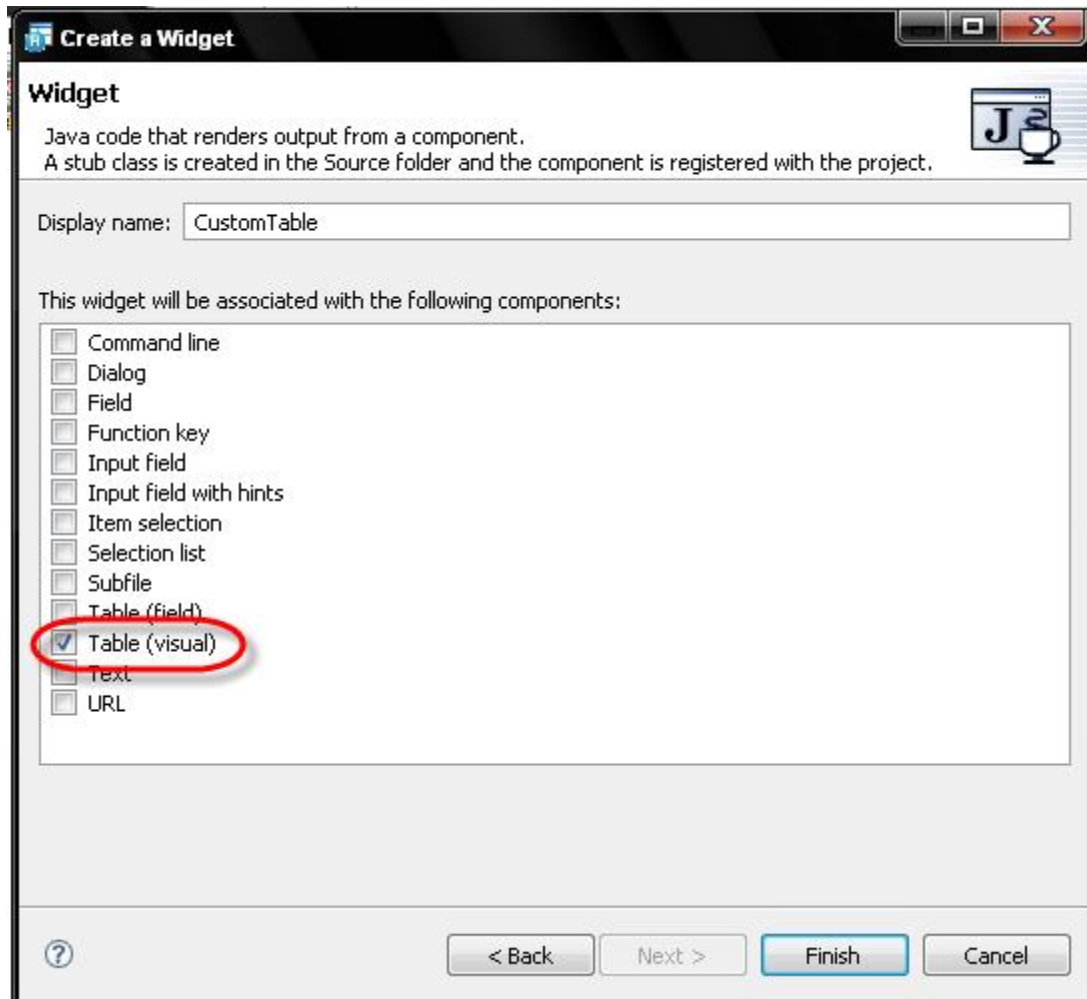
Our objective is to transform the above screen such that, the scrollable data is presented in a table, where each item is represented as a hyperlink. We use screen combination to collect all the data from the host to feed our custom widget, which in turn renders the data as shown below:

Source	Name	Org	Customer or Store #	Optional Data
02	%Dr. Smith	000	00000000000000000066	
01	Adam Smith	000	00000000000000000005	
E	Allen Smith	000	00000000000000000009	
E	Allen Smith	000	00000000000000000000	

As we can see in the above screen, our data from the host is displayed in a table in the transformation. The input fields on the host no longer appear on the webpage; instead each of the results on the webpage is clickable (hyperlink). Our custom widget incorporates all the functionality needed to render

the table and the functionality, such as populating the correct input field on the host, when its webpage counterpart is clicked.

We have associated our custom widget on the **Visual Table** component in HATS:



The **drawHTML** method in the class file of our widget is modified in order to render the host screen. We use a **TableCellComponentElement** object to contain all the data as cells, extracted from the screen combination (the event that fetches scrollable data on multiple screens from the host). This is shown in the code fragment seen below:

```
TableComponentElement tce=(TableComponentElement)elements[i];
TableCellComponentElement tcce [][] =tce.getCells();
buffer.append("<table>");
StringBuffer atagbuf = new StringBuffer(1024);
StringBuffer linklabelbuf = new StringBuffer(1024);
```

A **TableComponentElement** object named **tce** contains all the elements from the host screen. Next we create a **TableCellComponentElement** object named **tcce** that contains the extracted data as cells. This

allows us to treat the data as rows and columns. We can use iterations to print the data on the webpage. The following code segment depicts how the data can be printed out to the webpage:

```
linklabelbuf.append(" " +tcce[rc][cc].getText().trim());// this is the text the cursor hovers over
```

We can use different classes to highlight odd and even rows. This can be implemented by utilizing the code segment below:

```
if (oddeven.equals("HATSTABLEEVENROW")) {
    if (rowclass.equals("HATSTABLEODDROW")) {
        rowclass = "HATSTABLEEVENROW";
    } else {
        rowclass = "HATSTABLEODDROW";
    }
}
```

Our table appears as shown below:

Source	Name	Org	Customer or Store #	Optional Data
02	Dr. Smith	000	00000000000000000066	
01	Adam Smith	000	00000000000000000005	
E	Allen Smith	000	00000000000000000009	
E	Allen Smith	000	00000000000000000000	

Notice how the odd and even rows are highlighted using different classes.

When we render each row as a hyperlink, we have to keep in account of the input field at the start of each row. Whenever a row is clicked, an **X** needs to be populated in its corresponding input field followed by an **[enter]** to the host. For this to happen, it is important that we keep an accord of each and every input field. We can achieve this with the help of the code segment below:

```
for (int cc=0;cc<tcce[0].length;cc++){
    if (!tcce[rc][cc].isProtected())
    {
        int sp=tcce[rc][cc].getStartPos();
        atagbuf.append("<tr class='"+rowclass+"' align='left'><td align='left'>
        <input type='hidden' value='\" + \" \" name='in "+sp+" 1 "+tcce[rc][cc].getScreenId() +\" \" />"+
        "<a href='\"javascript:document.HATSForm.in "+sp+" 1 "+tcce[rc][cc].getScreenId()
        +\".value='X';ms(['enter'], 'HATSForm');\"><pre>\"";
```

In the above segment, we have used the **getStartPos** method to fetch the starting position of non protected fields (input fields). We also use the **getScreenId** function to fetch the screen ID number of every unique screen in the screen combination. We can use the starting position and screen ID form the complete name of our input fields such as **in_794_1_1**, where,

794 represents the starting position of a particular input field

1 represents the length of the input field (a 1 character input field will have 1, a 4 character will have 4 and so on). This 1 remains unchanged as all our input fields on the host are 1C only.

1 represents the screen id.

Upon further analyzing the code segment above, we notice that JavaScript has been used inside a hyper-reference which is responsible for populating the correct input field upon mouse click and sending the **[enter]** command to the host.

By following the above guidelines, we can successfully develop a table widget for the host screen scenario or a similar scenario that was under consideration in this article.



© Copyright IBM Corporation 2010
IBM Global Services
Route 100
Somers, NY 10589
U.S.A.
Produced in the United States of America
08-10
All Rights Reserved

IBM, the IBM logo, ibm.com, Lotus®, Rational®, Tivoli®, DB2® and WebSphere® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at ibm.com/legal/copytrade.shtml Other company, product and service names may be trademarks or service marks of others. The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided "as is" without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software. This document illustrates how one organization uses IBM products. Many factors have contributed to the results and benefits described; IBM does not guarantee comparable results elsewhere.